下载APP

07｜帧间预测：如何减少时间冗余？

2021-12-06 李江

《攻克视频技术》 课程介绍 >

讲述：李江

时长 35:14 大小 32.27M

你好，我是李江。

上一节课我们一起讨论了帧内预测。帧内预测主要是通过当前编码块它的相邻的已经编码完成的像素预测得到预测块，从而达到减少空间冗余的目的。我们知道在视频编码时主要需要去除 4 个冗余，包括：空间冗余、时间冗余、视觉冗余和信息熵冗余。那么这节课我们接着之前的课程，继续讲讲如何减少时间冗余。

提醒一下，这节课难度比较高，涉及的知识点又很多，建议你多看几遍。我也会尽量简化过程，把基本的原理讲清楚。你在这里打好了基础，再去学习更高阶的内容就轻松多了。

我们已知，视频在 1 秒钟内有很多帧图像，其通过帧率来表示。一般来说帧率为 24fps 或者 30fps，也就是指，1 秒钟会有多达 24 帧或者 30 帧图像。

但是其实在自然状态下，人或者物体的运动速度在 1 秒钟之内引起的画面变化并不大，且自然运动是连续的。所以**前后两帧图像往往变化比较小，这就是视频的时间相关性**。帧间预测就是利用这个特点来进行的。**通过在已经编码的帧里面找到一个块来预测待编码块的像素，从而达到减少时间冗余的目的。**

那么在正式讲解之前，还有三点我需要特别强调一下。

第一，在帧内预测中，我们是在当前编码的图像内寻找已编码块的像素作为参考像素计算预测块。而帧间预测是在其他已经编码的图像中去寻找参考像素块的。这正是**帧内预测和帧间预测的区别**。

第二，帧间预测是可以在多个已经编码的图像里面去寻找参考像素块的，我们称之为**多参考。多参考和单参考（只在一帧图像里面寻找参考像素块）其实底层的原理是一样的**，只是多参考需要多搜索几个参考图像去寻找参考块而已，所以**我们讲解的时候就使用单参考讲解。**这样既可以简化过程，你也可以更容易掌握帧间编码的基本原理。

第三，帧间预测既可以参考前面的图像也可以参考后面的图像（如果参考后面的图像，后面的图像需要提前先编码，然后再编码当前图像）。**只参考前面图像的帧我们称为前向参考帧，也叫 P 帧；参考后面的图像或者前面后面图像都参考的帧，我们称之为双向参考帧，也叫做 B 帧。**B 帧相比 P 帧主要是需要先编码后面的帧，并且 B 帧一个编码块可以有两个预测块，这两个预测块分别由两个参考帧预测得到，最后加权平均得到最终的预测块。**P 帧和 B 帧的底层逻辑基本是一样的。**同样为了简化过程，我们这节课**以 P 帧为例子来讲解底层逻辑**。

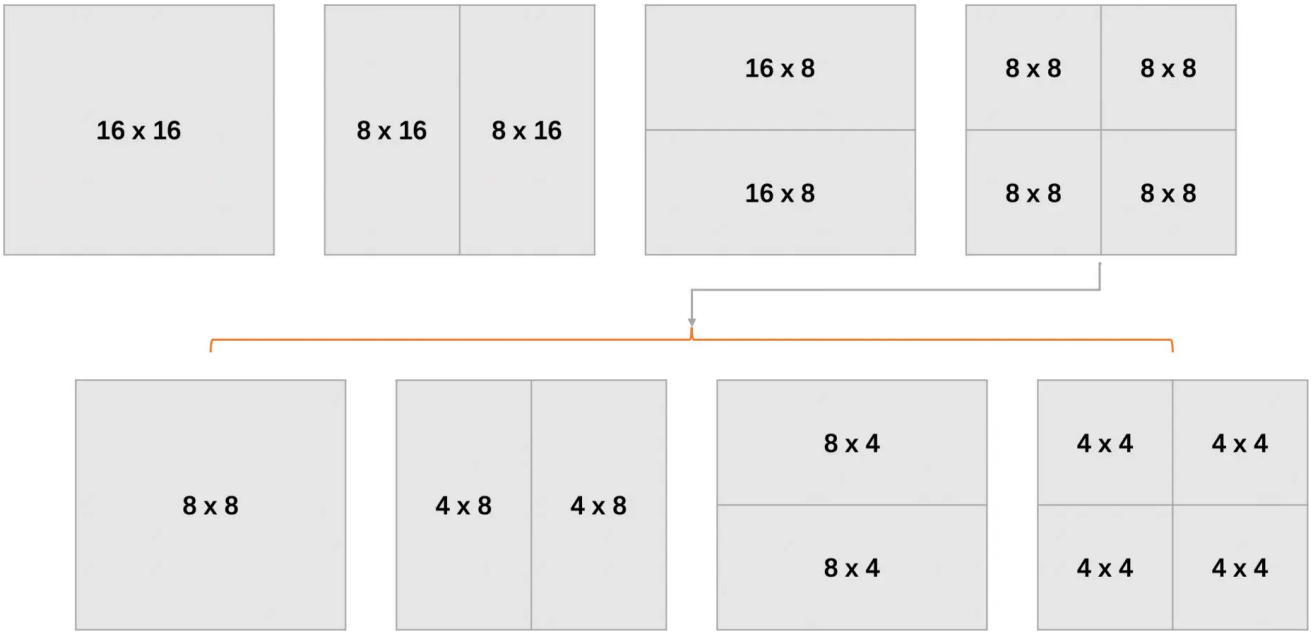
帧间编码

好了，接下来我们就以 H264 标准为基础来聊聊 P 帧的帧间编码过程吧。就从最基础的块大小开始讲起。

块大小

上节课，我们讲到了帧内预测有亮度 16 x 16、亮度 4 x 4 和色度 8 x 8 这几种块。类似地，在帧间预测也一样有不同的块和子块大小。**相比帧内预测，帧间预测的块划分类型要多很多。**宏块大小 16 x 16，可以划分为 16 x 8，8 x 16， 8 x 8 三种，其中 8 x 8 可以继

续划分成 8 x 4，4 x 8 和 4 x 4，这是亮度块的划分。在 YUV 4:2:0 中，色度块宽高大小都是亮度块的一半。亮度宏块的划分方式如下图所示：



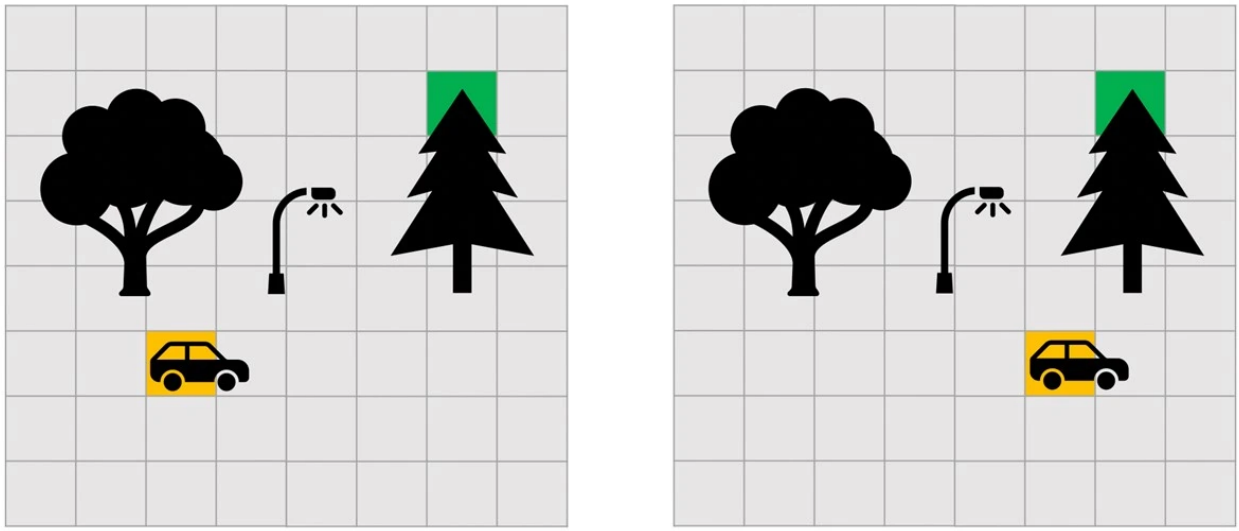
参考帧和运动矢量

在帧间预测中，我们会**在已经编码的帧里面找到一个块来作为预测块，这个已经编码的帧称之为参考帧**。在 H264 标准中，P 帧最多支持从 16 个参考帧中选出一个作为编码块的参考帧，但是同一个帧中的不同块可以选择不同的参考帧，这就是多参考。

通常在 RTC 场景中，比如 **WebRTC 中，P 帧中的所有块都参考同一个参考帧**。并且**一般会选择当前编码帧的前一帧来作为参考帧**。为什么呢？

这是因为自然界的运动一般是连续的，同时在短时间之内的变化相对比较小，所以前面的帧通常是最接近当前编码帧的，并且两者的差距比较小。因此，我们比较容易从前一帧中找到一个跟当前编码块差距很小的块作为预测块，这样编码块减去预测块得到的残差块的像素值很多都是 0，压缩效率是不是就很高了？

好了，选择好了参考帧之后，我们还有一个问题。虽然运动变化比较小，但是还是有变化啊，比如说下图中的场景。



1、绿色块对应的树梢是静止的不会动，因此运动矢量为(0, 0)

2、橙色块对应的小车向前行驶了3个宏块，因此运动矢量为(-16 * 3, 0) = (-48, 0)

极客时间

图中的小车在往前开，树是不动的。我们可以看到车相对于树的距离是变化的。那我们怎么来表示这个变化呢？

或者从编码的角度来讲，右边图像中橙色块的内容跟左边图像中橙色块的内容基本一样。很明显，如果以左边图像为参考帧的话，那么左边图像中的橙色块就是最合适的预测块，但是左右两幅图像橙色块的位置很明显又不同，而这种不同是因为图像中的小车的运动引起的。因此，为了表示这种变化，我们**用运动矢量来表示编码帧中编码块和参考帧中的预测块之间的位置的差值**。

比如说上面两幅图像中，小车从前一幅图像中的（32，80）的坐标位置，变化到当前图像（80，80）的位置，向前行驶了 48 个像素。很明显，如果我们选用（32，80）这个块作为当前（80，80）这个编码块的预测块的话，是不是就可以得到全为 0 像素的残差块了？这是因为小车本身是没有变化的，变化的只是小车的位置。

这个位置变化我们怎么表示呢？我们用运动矢量来表示。我们称（32 - 80， 80 - 80）也就是（-48， 0）为运动矢量。我们先**把运动矢量编码到码流当中**，这样解码端只要解码出运动矢量，使用运动矢量就可以在参考帧中找到预测块了，我们再解码出残差（如果有的话），残差块加上预测块就可以恢复出图像块了。

好了，这就是参考帧和运动矢量。可是还有一个问题：因为我们通过人眼能够看到小车在两幅图像的位置，所以我们可以在参考帧中找到一个与当前编码块相似的块作为预测块，

但是**编码器怎么找到这个预测块呢**？它们又没有眼睛。这就是运动搜索算法应该解决的问题啦。我们接下来就来聊聊运动搜索是怎么做的。

运动搜索

从前面的讨论我们知道，运动搜索的目标就是在参考帧中找到一个块，称之为预测块，且这个预测块与编码块的差距最小。从计算机的角度来说就是，编码块跟这个预测块的差值，也就是残差块的像素绝对值之和（下面我们用 SAD 表示残差块的像素绝对值之和）最小。

现在是不是目标就清晰很多了。比如说当前编码块大小是 16 x 16，那我们就先去参考帧中找到一个个 16 x 16 的块作为预测块，并用当前编码块减去预测块求得残差块，然后用我们经常做的绝对值求和操作得到两者之间的差距，最后选择差距最小的预测块作为最终的预测块。

所以，我们运动搜索的方法就很简单了，就是从参考帧中第一个像素开始，将一个个 16 x 16 大小的块都遍历一遍。我们总是可以找到差距最小的块。这种方法我们称之为全搜索算法。

全搜索算法一定可以搜索到最相似的预测块。但是你有没有发现这种方法有一个特别大的**缺点就是需要逐个像素去遍历每一个块，非常费时间**。由于帧间预测中每一个 16 x 16 的宏块还可以划分成上面讲的多种不同的子块大小，每一个子块也需要做一遍运动搜索。如果采用这种运动搜索算法的话，那编码一帧的时间将会非常长。

那有没有速度快一点的搜索算法呢？答案肯定是有的。下面我就来介绍一下常用的两种快速运动搜索算法。

在讲述快速算法之前我先说明一下，**搜索算法中每一个搜索的点都是搜索块的左上角像素点**。比如说菱形和六边形搜索的几个点都指的是以该点为左上角像素点的块。如果所搜块大小为 16x16，则指的是以该点为左上角像素点的 16x16 的块。

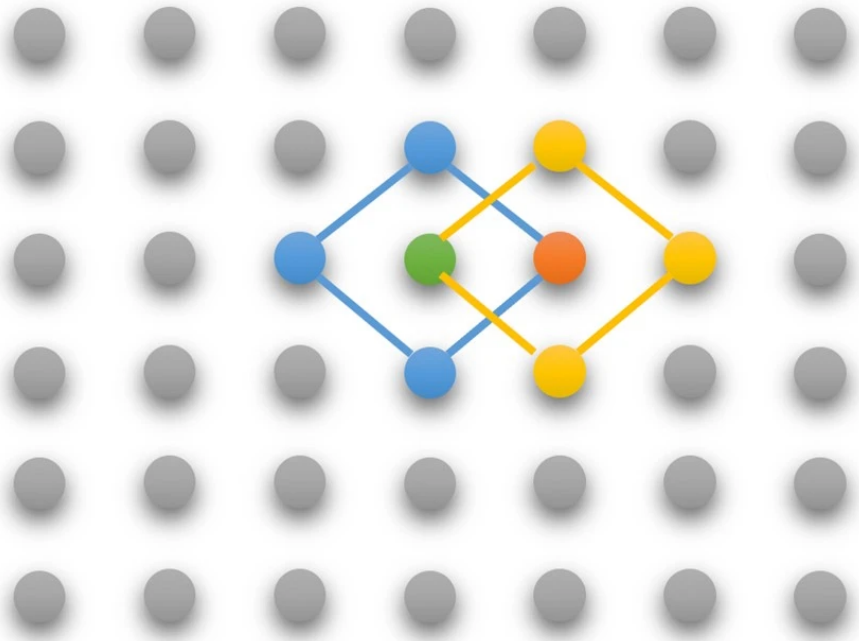
1. 钻石搜索算法

   钻石搜索算法又称为菱形搜索算法。顾名思义，就是**以一个菱形的模式去寻找最优预测块**。这里我们以亮度 16 x 16 的块的运动搜索为例，将算法的思想分为以下几个步骤：

（1）从搜索的起始点开始，以起始点作为菱形的中心点。首先以该中心点为左上角像素的 16 x 16 的块作为预测块，求得残差块并求得像素绝对值之和，也就是 SAD。之后对菱形 4 个角的 4 个点分别做同样的操作求得 SAD 值。得到最小的 SAD 值，最小 SAD 值对应的点就是当前最佳匹配点。

（2）如果最佳匹配点是菱形的中心点，那我们就找到了预测块了，搜索结束。

（3）如果最佳匹配点不是菱形的中心点，则用以当前最佳匹配点为中心点的菱形继续搜索，重复之前的步骤直到菱形的中心点为最佳匹配点。



例如，上图中：

第一步，以绿色点（起点）为中心点，搜索绿色点和旁边蓝色线连接的 4 个点，得到的最佳匹配点为橙色点，非中心点。

第二步，再以橙色点为中心点，搜索橙色点和旁边黄色线连接的 4 个点，最佳匹配点是中心点橙色点，搜索完毕，橙色点为最佳匹配点。

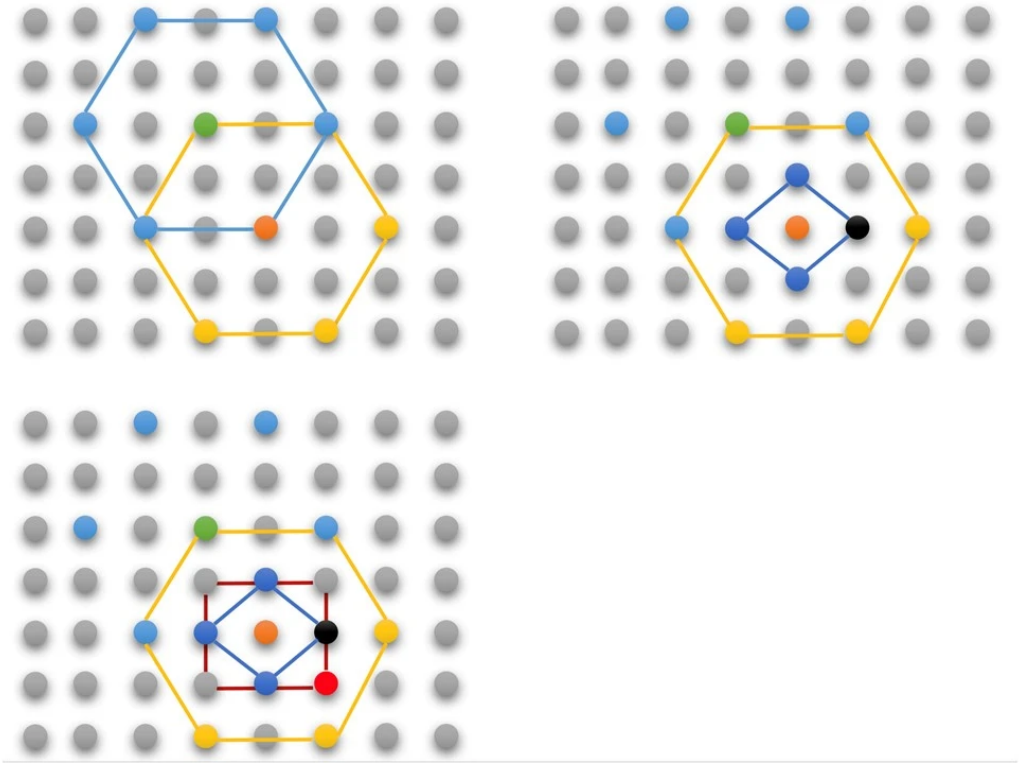
2. 六边形搜索算法

六边形搜索跟钻石搜索差不多，只是搜索模式是六边形的。我们还是以亮度 16 x 16 的块为例来介绍一下，主要有下面几个步骤：

（1）从搜索的起始点开始，以起始点作为六边形的中心点。求得中心点作为左上角像素的预测块的 SAD 值。之后对六边形的角上的 6 个点做同样的操作求得 SAD 值。得到最小的 SAD 值，而最小 SAD 值对应的点就是当前最佳匹配点。

（2）如果最佳匹配点是六边形的中心点，那我们就用以该点为中心点的菱形和正方形各进行一次精细化搜索。找到中心点、菱形的 4 个顶点和正方形 4 个顶点中 SAD 最小的点作为最佳匹配点。

（3）如果最佳匹配点不是六边形的中心点，则用以当前最佳匹配点为中心点的六边形继续搜索，重复之前的步骤直到中心点为最佳匹配点。



例如，上图中：

第一步，以绿色点（起点）为中心点，搜索中心点和旁边蓝色线连接的 6 个点，得到的最佳匹配点为橙色点，非中心点。

第二步，再以橙色点为中心点，搜索橙色点和旁边黄色线连接的 6 个点，最佳匹配点是是中心点橙色点。

第三步，再以橙色点为中心点，搜索橙色点和旁边蓝色线连接菱形的 4 个点，最佳匹配点为黑色点。

第四步，还是以橙色点为中心点，搜索旁边红色线连接的正方形的 4 个点，并与菱形搜索得到的最佳匹配点黑色点比较，找到最后的最佳匹配点为红色点，搜索完毕。

通过上面的快速搜索算法我们就能够得到编码块在参考帧中的最佳匹配点，**以最佳匹配点为左上角像素的块就是预测块**，并且**预测块左上角像素在参考帧中的坐标 (x1, y1) 与编码块在当前编码帧中的坐标 (x0, y0) 的差值（x1 - x0, y1 - y0）就是运动矢量**。

上面两种运动搜索算法都是以搜索形状的中心点为最佳匹配点结束的。但是还有一个问题我们没有解释，那就是搜索的起始点怎么确定呢？

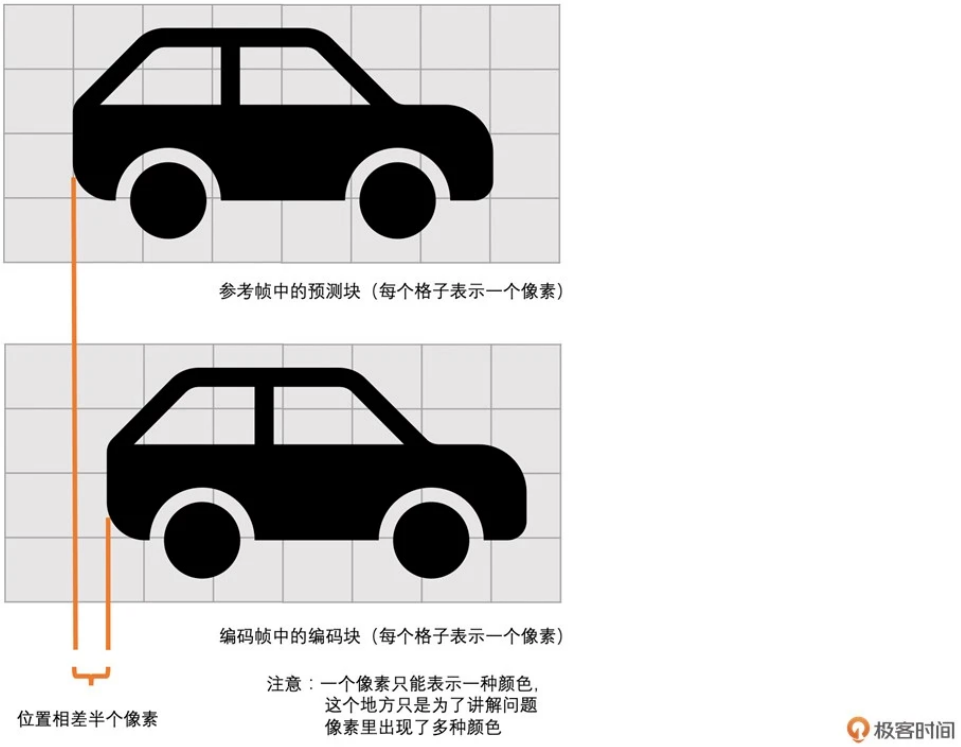
其实搜索的起始点可以使用当前编码块的左边块、右边块、左上角块和右上角块的运动矢量预测得到。具体预测方法我们会在下面的运动矢量预测小节里面讲解。其总体的思路就是我们认为，一般一个块最大也就 16 x 16 的大小，而运动的物体一般远大于这个大小，所以相邻块的运动方向大多数是很相似的。因此，**我们一般会通过相邻已经编码块的运动矢量来预测当前块的运动矢量。这个预测的运动矢量也经常用做搜索的起点。**

有了快速运动搜索算法我们就不需要遍历整个参考帧的像素去寻找预测块了，这样速度可以快很多。但是必须要说明一下，就是快速搜索算法也有一个缺点，**它搜索到的预测块不一定是全局最优预测块**，也就是说不一定是最相似的块，有可能是局部最优预测块。

但是实验数据表明，快速搜索算法相比全搜索算法压缩性能下降非常小，速度却可以提升十几倍到几十倍。所以总的来说，我们可以认为**快速搜索算法是远好于全搜索算法的**，并且一般全搜索算法是不会实际使用的。

好了，经过上面的讨论，我们已经知道了运动搜索的具体思路了。但是我们还有一个疑问：如果一个物体运动了，比如小车向前行驶了 48 个像素点，那我们就可以通过运动矢量（-48，0）在参考帧中找到小车。我们前面也说了，小车的运动是连续的，**如果小车向前行驶了 48.5 个像素点呢？又或者是向前行驶了 48.25 个像素点呢？**运动矢量选择（-48.5， 0）或者（-48.25，0）吗？可是 0.5 个像素点是什么样的，0.25 个像素点又是什么样的？图像上都没有这种像素点啊，怎么办呢？

其实没关系的，我们还是可以使用（-48，0）作为运动矢量，只是预测块中的小车位置与我们编码块中的小车位置会相差个 0.5 或者 0.25 个像素，得到的残差会大一些，压缩效率稍微低一些，问题也不大。



比如上面图中的小车，如果直接用编码块减去参考块的话，因为两个块中小车的位置没有完全重合，所以残差块不等于 0。这样还是有残差的，因此压缩效率会低一些。

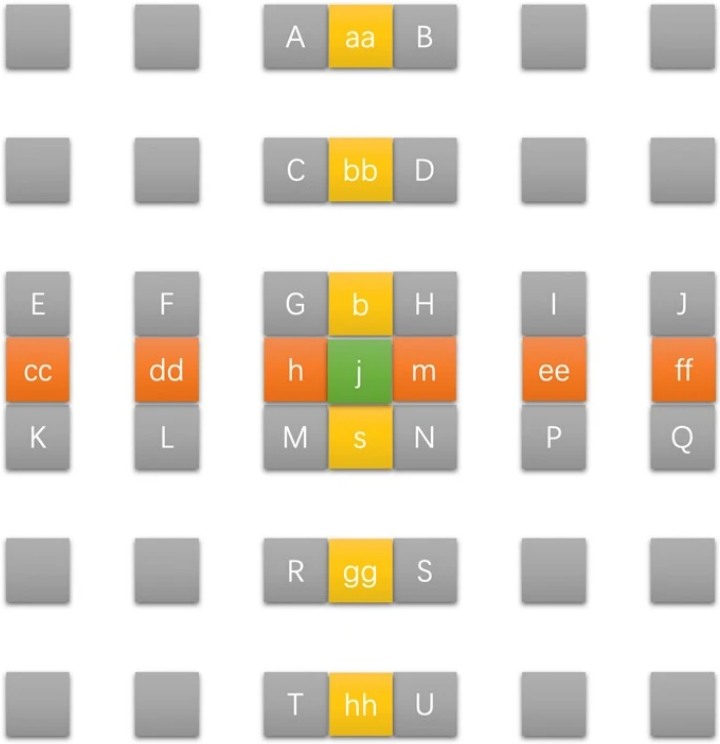
但是勤劳智慧的人类是不会停止前进的脚步的。为了能够解决这种半个像素或者 1/4 个像素的运动带来的**压缩效率下降**的问题，我们通过对参考帧进行半像素和 1/4 像素插值（统称为亚像素插值）的方式来解决。

什么意思呢？就是**我们用插值的方式将半像素和 1/4 像素算出来，也当作一个像素**，这样小车向前行驶 48.5 个像素也好，向前行驶 48.25 个像素也好，都是可以通过运动矢量找到比较准确的位置的。那亚像素插值具体怎么做呢？我们接下来看一下。

亚像素插值

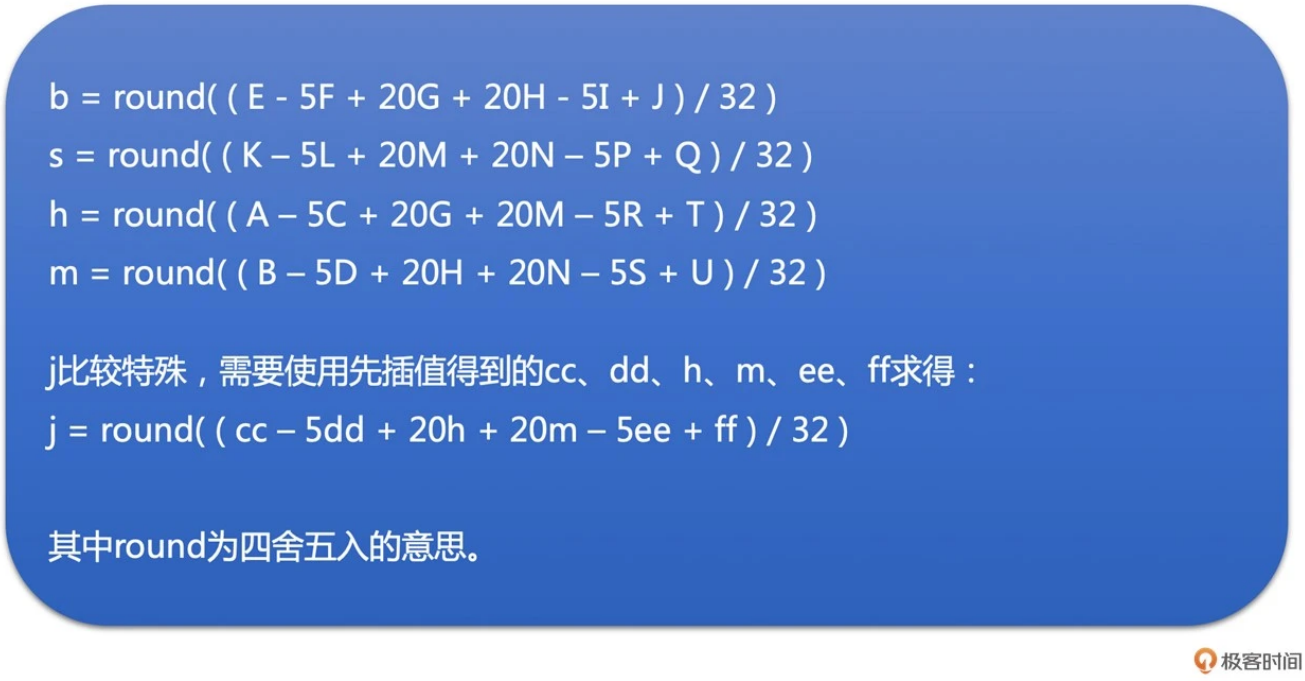
亚像素插值归根到底还是插值操作，我们在第 3 节课里面已经讨论过了一些插值算法。亚像素插值的思想跟前面课里的插值算法的思想是一样的，都是通过已经有的像素点经过一定的加权计算得到需要求得的像素。

在这里，已经有的像素就是整像素（就是图像本身有的像素称为整像素），需要插值求得的就是半像素和 1/4 像素。其中半像素通过整像素插值得到，1/4 像素又是通过整像素和半像素插值得到的。因此，**我们先通过整像素插值得到半像素，然后再通过半像素和整像素插值得到 1/4 像素**。半像素的插值过程可以通过下面的图示表示：

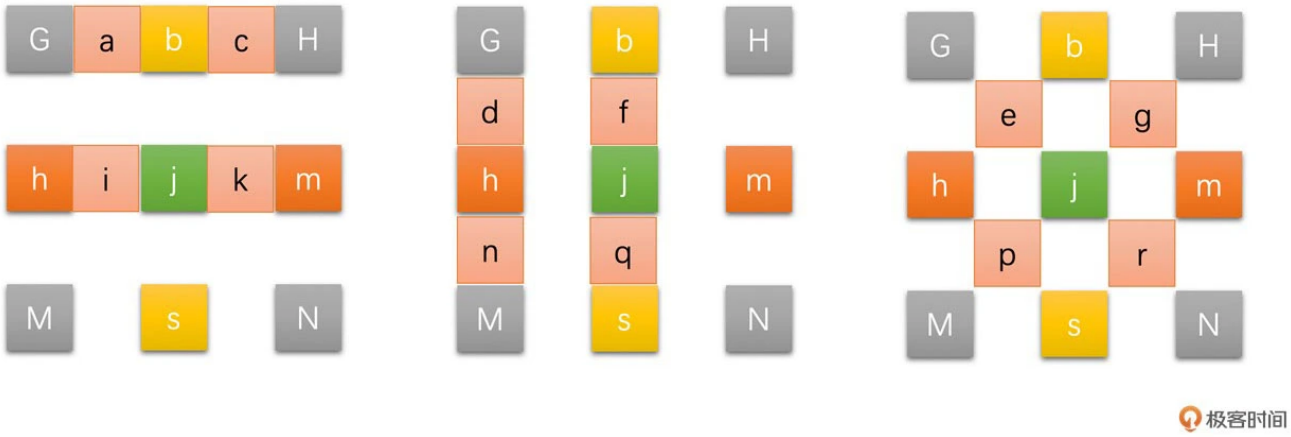


其中，灰色为整像素点，橙色为水平半像素，黄色为垂直半像素点，绿色为中心半像素点。

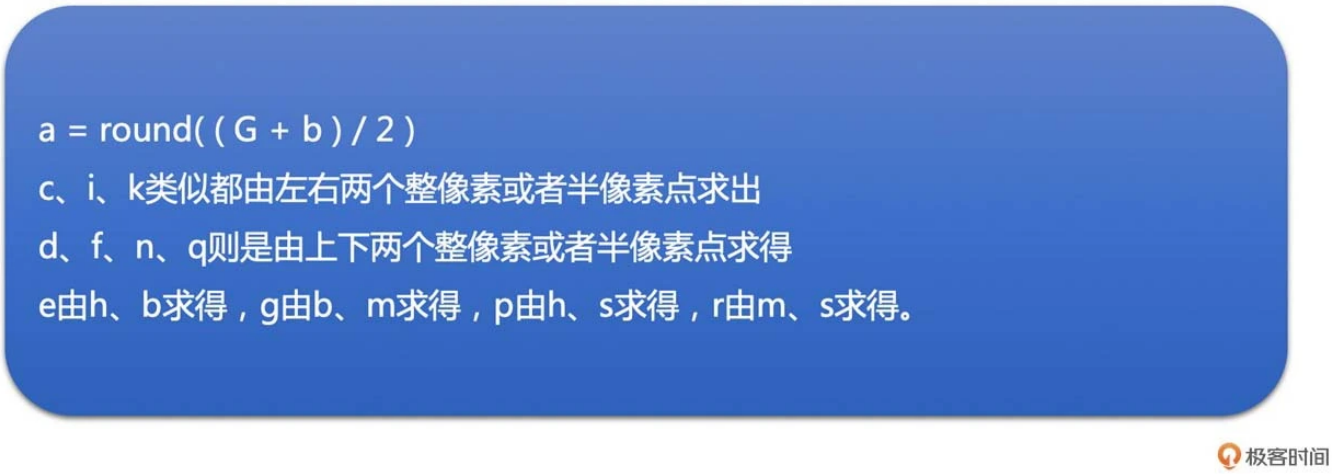
半像素点的插值是以 6 个整像素点使用六抽头插值滤波器计算得到的，滤波器权重系数为：(1/32, -5/32, 5/8, 5/8, -5/32, 1/32)。具体计算方法如下：



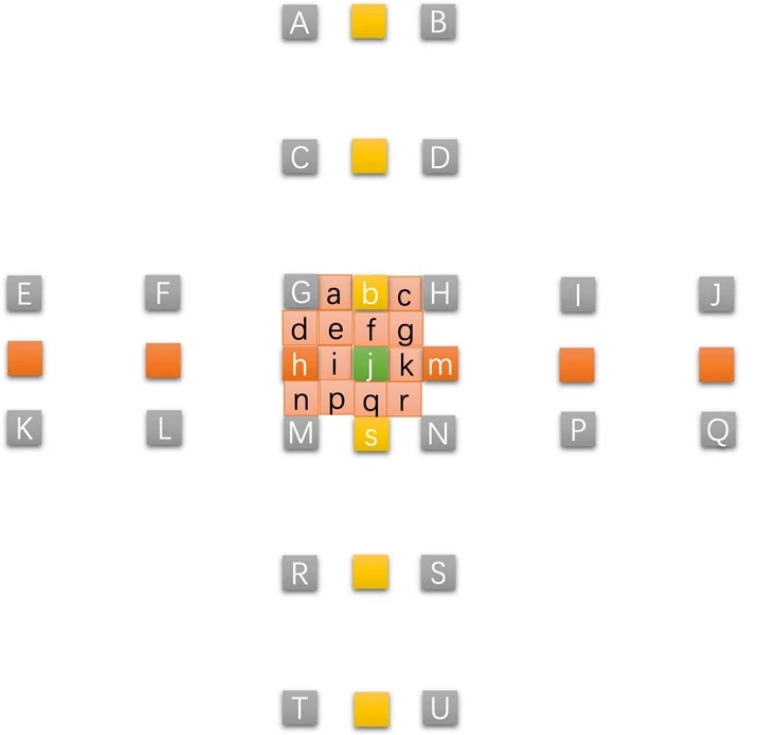
得到了半像素之后，1/4 像素就比较简单，由整像素和半像素求平均值得到，其插值过程可以通过下图表示：



其中，红色点为 1/4 像素点，具体计算方法如下：



整个半像素和 1/4 像素的插值过程可以通过下图表示：



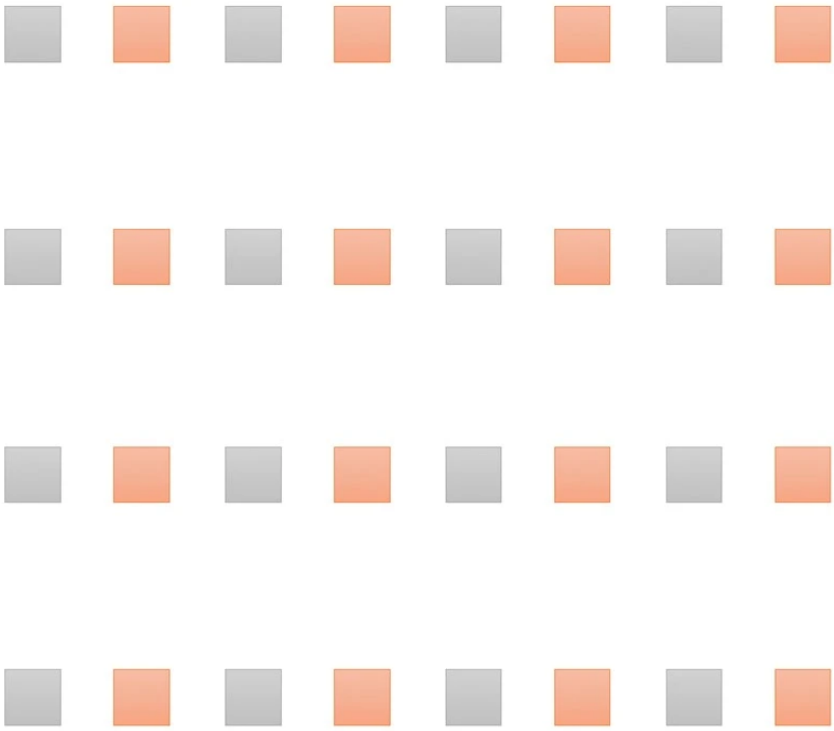
极客时间

插值得到了所有的半像素和 1/4 像素之后，我们就可以用运动矢量表示 0.5 个像素和 0.25 个像素的移动了，就如之前小车向前行驶了 48.5 个像素。如果只是整像素的图像，那只能表示出向前行驶了 48 个像素或者 49 个像素，没办法表示 48.5 个像素。

我们用下面的几幅图像来直观地感受一下一个 4x4 的像素块的半像素图像和 1/4 像素图像的像素位置。首先来看一幅 4x4 的像素块的像素位置。

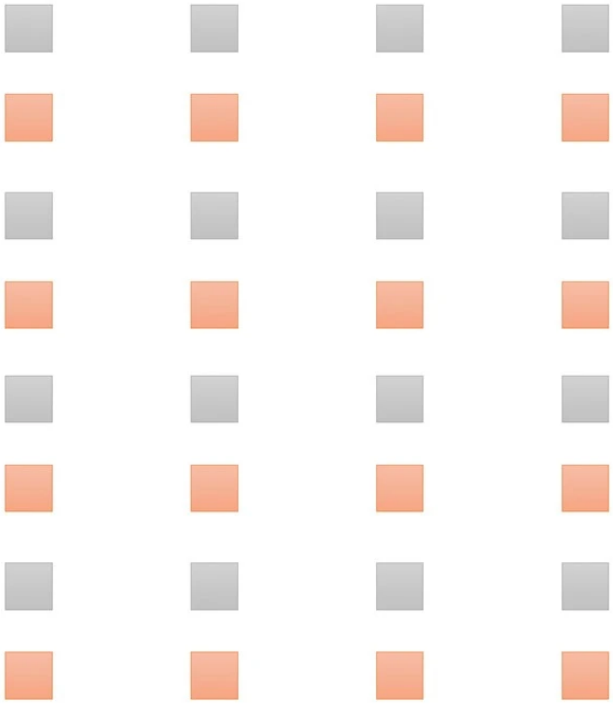
极客时间

下面图中的红色的 4x4 像素表示水平半像素图像。



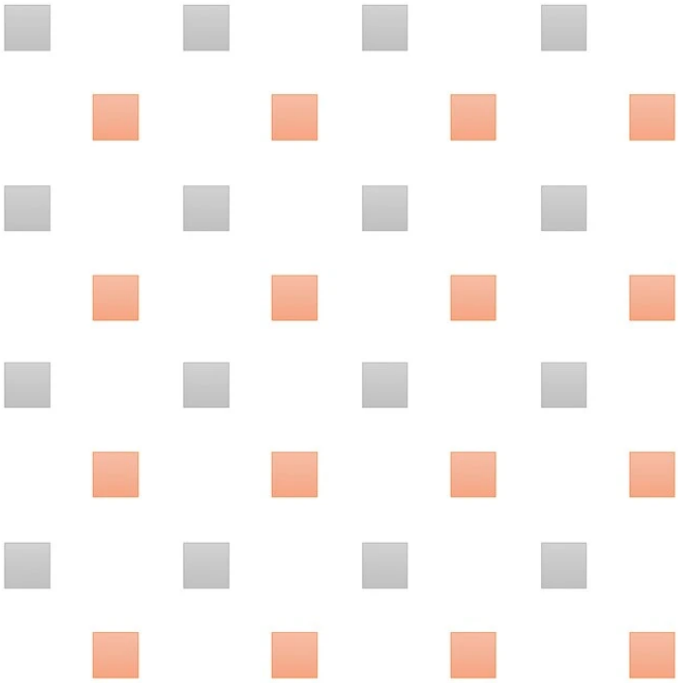
极客时间

下面图中的红色的 4x4 像素表示垂直半像素图像。



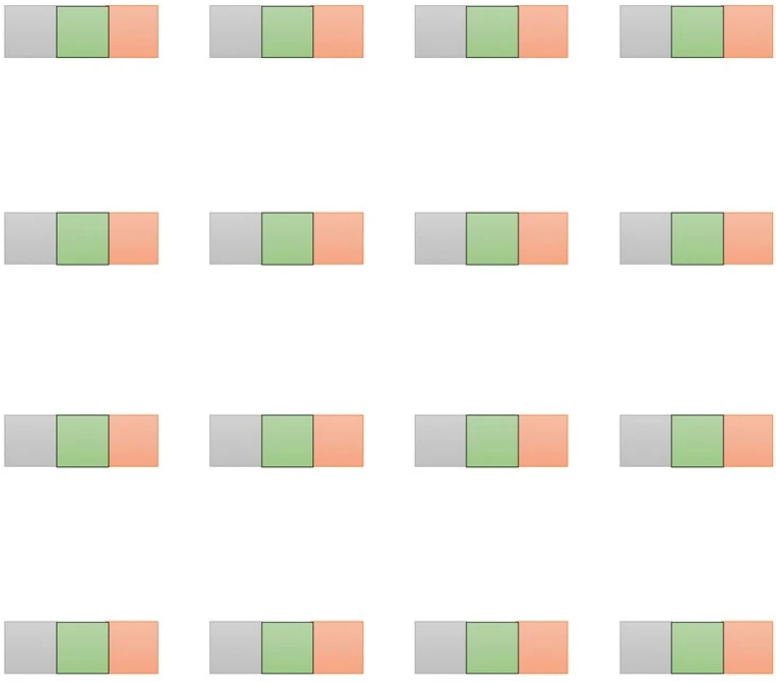
极客时间

下面图中的红色的 4x4 像素表示水平、垂直半像素图像。



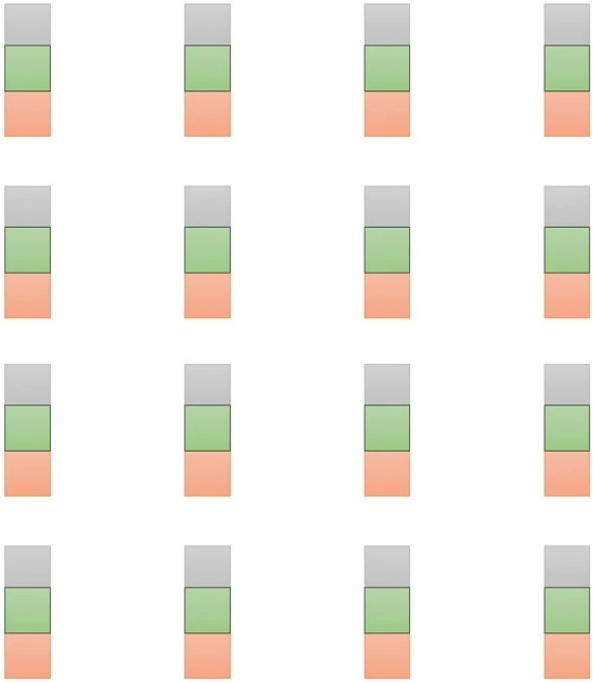
极客时间

下面图中的绿色的 4x4 像素表示水平 1/4 像素图像。



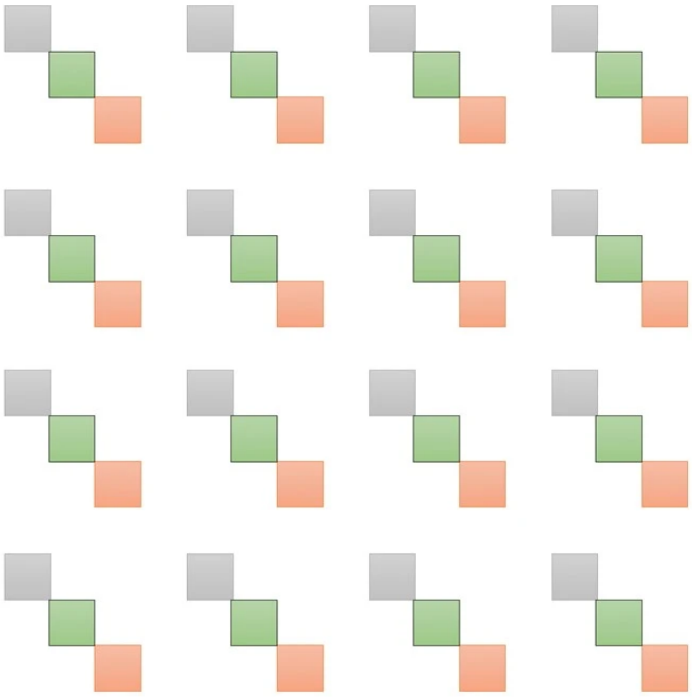
极客时间

下面图中的绿色的 4x4 像素表示垂直 1/4 像素图像。



极客时间

下面图中的绿色的 4x4 像素表示水平、垂直 1/4 像素图像。



极客时间

有了上面的亚像素插值算法，前面的小车运行 48.5 个像素的问题就可以通过半像素插值在参考帧中插值得到一辆新的小车，从而就可以解决之前预测块和编码块位置不重合的问题。不过，还需要说明一下的就是，**插值得到的小车跟原始的小车的对应像素点的像素值并不是完全一样的**，毕竟插值得到的像素点是利用滤波算法加权平均得到的。

因此，半像素插值得到的预测块并不一定就比整像素预测块的残差小。只是我们多了很多个半像素预测块和 1/4 像素预测块的选择，所以**我们可以在整像素预测块、半像素预测块和 1/4 像素预测块里面选择一个最好的**。怎么选择呢？其实是**在整像素运动搜索的基础上，再做一次精细化的亚像素运动搜索**。下面我们就来讲讲亚像素运动搜索的过程是怎样的。

亚像素精度运动搜索

有了插值得到的亚像素，我们就可以进行亚像素精度的搜索了。一般搜索算法步骤如下：

1. 先通过快速搜索算法进行整像素运动搜索算法得到整像素的运动矢量（就是我们在运动搜索小节中讲述的内容）。
2. 对参考帧进行半像素和 1/4 像素插值。
3. 以整像素运动矢量指向的整像素为起点，进行钻石搜索算法，分别求得中心点以及上、下、左、右四个半像素点对应预测块的残差块，得到 SAD 值。取 SAD 值最小的点为最佳匹配点。
4. 以半像素运动搜索的最佳匹配点为起点，分别求得中心点以及上、下、左、右四个 1/4 像素点对应预测块的残差块，得到 SAD 值，并取最小的点为最佳匹配点。
   通过上面亚像素搜索算法得到的最佳匹配点就可以得到最后的运动矢量了。假设整像素运动矢量为 (a0, b0)，半像素最佳匹配点相对于整像素最佳匹配点的运动矢量为 (a1, b1)，1/4 像素最佳匹配点相对于半像素最佳匹配点的运动矢量为 (a2, b2)，则最后运动矢量（a，b）的值的计算方法如下：

$$(a, b) = (a0 << 2 + a1 << 1 + a2,\ b0 << 2 + b1 << 1 + b2)$$

极客时间

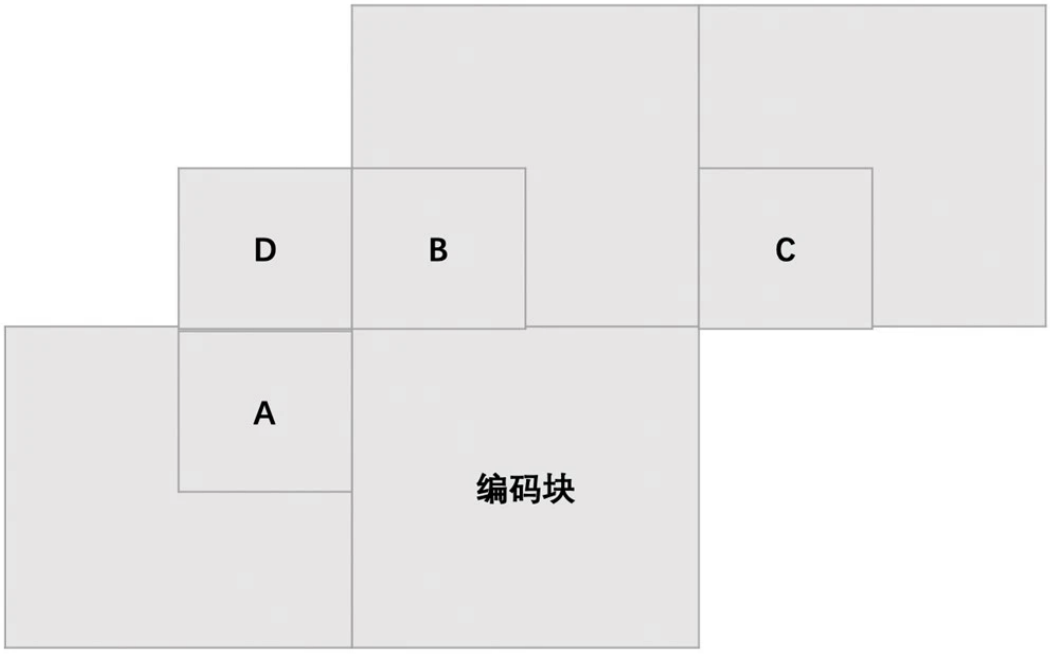
相当于原先的运动矢量乘以了 4，即原先 1/4 像素的 0.25 变成了 1，0.5 像素变成了 2，1 个像素则变成了 4。这主要是因为我们不用小数形式来表示运动矢量。因为浮点型数据会有精度误差，所以我们通过乘以 4 把它变成整数。

通过上面的整像素运动搜索和亚像素精度运动搜索，我们就得到了最终的运动矢量了。有了运动矢量之后，我们需要将运动矢量的信息也编码到码流中，并且解码的时候直接取出来用就可以在参考帧中把预测块找出来了。那运动矢量是直接编码到码流中的吗？其实不是的。那是怎么做的呢？接下来我们就来讨论一下运动矢量的预测。

运动矢量预测

其实，运动矢量跟我们的编码块一样不是直接编码进去的，而是先用周围相邻块的运动矢量预测一个预测运动矢量，称为 MVP。将当前运动矢量与 MVP 的残差称之为 MVD，然后编码到码流中去的。解码端使用同样的运动矢量预测算法得到 MVP，并从码流中解码出运动矢量残差 MVD，**MVP+MVD 就是运动矢量了**。

那运动矢量预测算法是怎样的呢？我们以 16 x 16 宏块为例通过下图来描述。



极客时间

其步骤如下：

1. 取当前编码宏块的左边块 A、上边块 B、右上块 C。如果右上块不存在或者参考帧与当前编码宏块不同（多参考的时候会存在），则使用左上块 D 替换 C，即 C = D。
2. 求得 A、B、C 块的参考帧有多少个与当前编码块的参考帧相同，记为 count。
3. 如果 count > 1，则取 A、B、C 块的运动矢量的中值（就是 A、B、C 块运动矢量的 3 个 x 和 3 个 y 分别取中间值作为 MVP 的 x 和 y）。

4. 如果 count = 1，则直接将这个块的运动矢量作为 MVP。
5. 如果 count = 0，并且 B、C 都不存在，A 存在的话，则直接将 A 的运动矢量作为 MVP。
6. 如果上述条件都不满足，则取 A、B、C 块运动矢量的中值。

SKIP 模式

了解了运动矢量的预测算法之后，我们还需要介绍一种比较特殊的模式，也是经常会遇到的一种帧间模式，就是 SKIP 模式。它的定义是这样的，如果运动矢量就是 MVP，也就是说 MVD 为 (0，0)，同时，残差块经过变换量化后系数也都是等于 0，那么当前编码块的模式就是 SKIP。

相比于 SKIP 模式，其它模式要不就是 MVD 不为 0，要不就是量化后的残差系数不为 0，或者两者都不为 0。所以说 **SKIP 模式是一种特例，由于 MVD 和残差块都是等于 0，因此压缩效率特别高**。

比如说 P 帧中的静止部分，前后两帧不会变化，运动矢量直接为 0，而且残差块像素值本身因为几乎没有变化基本为 0，只有少部分噪声引起的比较小的值，量化后更是全部变成了 0。**这种图像中的静止部分或者是图像中的背景部分大多数时候都是 SKIP 模式。**这种模式非常省码率，且压缩效率非常高。因为需要编码的信息非常少，所以单独在这里跟你讨论一下。

好了，到这里基本的帧间编码的知识我们都有了，接下来我们再来总体过一下 P 帧宏块的模式选择的过程。

帧间模式的选择

编码块帧间模式的选择其实就是参考帧的选择、运动矢量的确定，以及块大小（也就是块划分的方式）的选择，如果 SKIP 单独拿出来算的话就再加上一个判断是不是 SKIP 模式。我们主要是确定这 4 个东西。

之前的讨论当中我们都是以当前编码帧的前一帧作为参考帧的，也就是说是单参考的，不涉及到参考帧的选择。其实，如果是多参考的话，编码块在选择参考帧的时候只需要遍历每一个参考帧进行块划分，然后再对每一个块进行运动搜索得到运动矢量就可以了。跟单

参考相比就是多了一个参考帧遍历的操作。所以**我们这里还是以单参考帧的方式来讲讲帧间模式的选择过程**。

注意，帧间模式的选择大多数是看编码器的实现的，并且不同编码器实现都会不一样，所以我们只是讲讲其中一种模式选择的思路，具体的细节各个编码器都各不相同。具体选择过程如下：

1. 首先判断当前宏块是不是可以作为 SKIP 块（通过相邻已经编码的块是不是存在 SKIP 块，和当前块使用 MVP 做运动矢量之后，残差块变换量化后是不是都为 0 等算法来判断），如果可以作为 SKIP 块则模式选择结束，不再进行下面的划分了。
2. 宏块大小为 16 x 16。首先不划分宏块，直接使用 16 x 16 大小的块，在参考帧中进行运动搜索，得到运动矢量和预测块，通过 MVP 求得 MVD，通过预测块求得残差块，并求得残差块的 SATD 值（残差块经过 Hadamard 变换之后求绝对值再求和），估计 MVD 的编码后占用的字节数，将两个值加起来作为 cost16x16。
3. 将 16 x 16 块划分成 4 个 8 x 8 的子块，分别进行运动搜索，并求得每一个 8 x 8 子块的 MVD 和残差块，最后分别得到 4 个子块的 cost8x8。

（1）如果 4 个 8 x 8 子块的 cost8x8 之和小于 16 x 16 块的 cost16x16 的话，我们再分别对每一个 8 x 8 子块划分成 4 个 4 x 4 子块，同样分别进行运动搜索，得到每一个 4 x 4 子块的 cost4x4。

如果 4 个 cost4x4 之和小于 cost8x8，则将 8 x 8 块划分成 4 x 8 和 8 x 4 两种子块分别求得 cost4x8 和 cost8x4，再根据 4 个 cost4x4、2 个 cost4x8 和 2 个 cost8x4 的大小，选择最终的 8x8 划分的方式，并将对应的 cost 值更新到 cost8x8。

否则不划分 8 x 8 子块。

（2）如果 4 个 8 x 8 子块的最新的 cost8x8 之和还是小于 cost16x16 的话，则再将 16 x 16 划分成两个 8 x 16 和 16 x 8 子块，并分别求得 cost8x16 和 cost16x8，对比 8x8、16x8、8x16 的 cost 值，并决定最终 16 x 16 块的划分方式。

（3）否则的话，不划分 16 x 16 的块。

4. 得到了编码宏块的帧间模式之后，我们还需要对编码宏块进行帧内模式的选择。是的，没错。在 P 帧和 B 帧中的宏块也是可以使用帧内模式的，所以我们需要看是帧间模式 cost 更小还是帧内模式 cost 更小。这也回答了我们在第 4 节课里留的思考题。如果帧内模式更小则使用帧内模式；如果是帧间模式更小则使用帧间模式。但是一般来说 P 帧和 B 帧宏块决策出来绝大多数还是帧间模式的。

小结

好了，这节课到这里就要结束了。我们现在来回顾一下这节课的知识点。

我们通过一步步分析如何在参考帧中准确地找到预测块的方式，讲解了帧间预测中最重要的一些知识点。其主要包括以下 5 个方面：

1. 宏块的划分。**为了能够更准确的找到预测块，我们可以将 16 x 16 的宏块继续划分成更小的子块来做运动搜索**。因为图像有的地方静止的背景画面或者平坦的区域可以直接选用最大的块来搜索预测块；而有的地方细节很多，图像中的物体运动方向也各不相同，可能就需要划分成更小的块来做运动搜索。这样每一个块都拥有自己独立的运动矢量，并且得到的预测块更接近于编码块，从而有利于提高压缩效率。
2. 参考帧和运动矢量。**在 RTC 场景中我们一般选择单参考，并且一般选择当前编码图像的前一帧作为参考帧。**运动矢量是用来表示参考帧中预测块与编码帧中编码块位置的相对距离的。
3. 运动搜索。运动矢量是通过运动搜索得到的，而运动搜索是在参考帧中进行的。通常我们会使用钻石搜索和六边形搜索等快速运动搜索算法。一般不会使用全搜索算法。其中钻石搜索算法更简单，步骤更少，所以**如果需要编码速度快，一般选择钻石搜索**。六边形搜索步骤更多，更精细，**要求编码质量高，同时对速度要求不高的时候，可以选择六边形搜索**。
4. 亚像素插值和亚像素精度搜索。光做整像素运动搜索不太能够准确的处理连续运动场景。为了能够处理好这种连续运动的问题，我们对参考帧进行亚像素插值得到半像素和 1/4 像素图像。然后在整像素搜索的基础上在亚像素图像上做亚像素精度的运动搜索。实验数据证明，**半像素和 1/4 像素精度的运动搜索相比整像素精度的运动搜索可以明显地提高压缩效率**。
5. 在最后我们大体讲了一下编码块帧间预测模式的具体选择过程，并单独讲解了一下 SKIP 模式。**SKIP 模式是一种比较特殊的模式，由于 MVD 和残差块都是等于 0，因此其压缩**

效率特别高。

最后，我再次强调一下，今天的课程内容非常多。你需要多看几遍才能有比较好的理解。如果你是视频编码的应用工程师的话，掌握其中的思想就可以了；如果你是视频编码算法工程师的话可以先阅读几遍文章，之后再通过阅读编码器具体代码来熟练掌握。实际上，真正的帧间预测过程要比我讲述的内容复杂的多，有很多繁琐的细节，同时每一个编码器的具体实现也不一样，需要你自己课后慢慢学习。

参考文献：

https://blog.csdn.net/leixiaohua1020/article/details/45936267。

这篇文章中雷霄骅大神针对 H264 的帧间预测原理和 x264 中对应的代码做了非常详细地讲解。在这里非常感谢雷神对视频技术做出的贡献，以及无私地分享视频技术知识。雷神也是我学习视频技术路上的导师。在这里向雷神致敬。

思考题

学完了这节课，我有一个思考题留给你。P 帧的第一个宏块的 MVP 怎么得到呢？

欢迎你在留言区和我分享你的思考和疑惑，你也可以把今天所学分享给身边的朋友，邀请他加入探讨，共同进步。下节课再见。

分享给需要的人，Ta订阅后你可得 20 元现金奖励

生成海报并分享

赞 0 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 06 | 帧内预测：如何减少空间冗余？

下一篇 08 | 变换量化：如何减少视觉冗余？

精选留言 (3)

写留言

***佳**

2021-12-08

P帧B帧可以进行帧内预测吗？每一帧都会进行帧内和帧间两个预测吗？

Geek_7de4c5

2021-12-06

可惜雷神离开我们很久了。

展开

1

晓龙

2021-12-06

先问一个问题，P，B帧是帧内帧间算法二选一么？我一直以为是先帧间，然后帧内，两次压缩。为什么不选用两次压缩呢？是 压缩时间/压缩效果 太大了么？

P 帧的第一个宏块的 MVP 怎么得到呢？
MVP等于自身块的运动矢量，MVD = （0，0）

展开